Outline for today

- Differentiable threshold units
- Backpropagation algorithm
- Hidden layer representations

Perceptrons and Linear Units: Summary

Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

Linear unit training rule uses gradient descent

- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data not describable in H

Gradient Descent

Do until satisfied

1. Compute the gradient $\nabla E[\vec{w}]$, where

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

2. $\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$, where

$$\Delta\vec{w} = -\eta \nabla E[\vec{w}]$$

and η is a small constant

Properties:

- Each iteration reduces training examples error E
- Converges to local minimum (if η sufficiently small)
- Each iteration considers every example

Incremental Gradient Descent

Batch mode Gradient Descent:

Do until satisfied

1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

Incremental mode Gradient Descent:

Do until satisfied

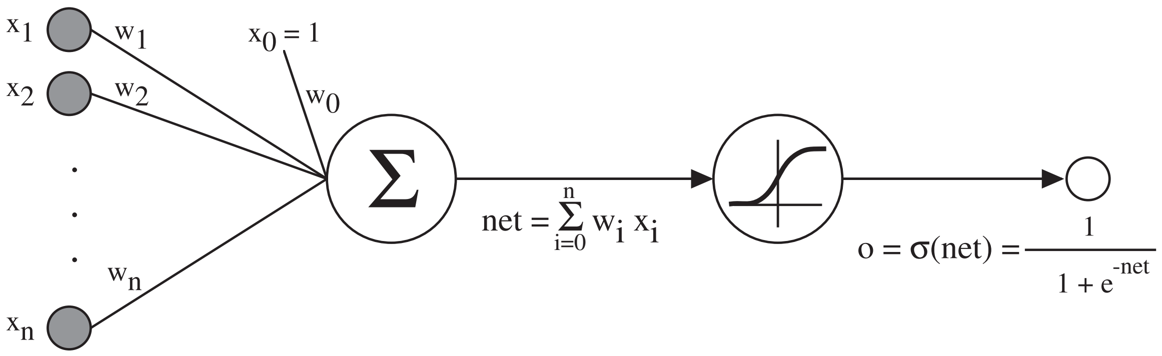
- For each training example p in D
  1. Compute the gradient $\nabla E_p[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_p[\vec{w}]$

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{p \in D} (t_p - o_p)^2$$

$$E_p[\vec{w}] \equiv \frac{1}{2}(t_p - o_p)^2$$

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough

Sigmoid Unit



$\sigma(x)$ is the sigmoid function $\frac{1}{1+e^{-x}}$

$\sigma(x)$ called the *squashing function*

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(X))$

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units $\rightarrow$ Backpropagation

Error Gradient for a Sigmoid Unit

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{p \in D} (t_p - o_p)^2 \\
&= \frac{1}{2} \sum_p \frac{\partial}{\partial w_i} (t_p - o_p)^2 \\
&= \frac{1}{2} \sum_p 2(t_p - o_p) \frac{\partial}{\partial w_i} (t_p - o_p) \\
&= \sum_p (t_p - o_p) \left(- \frac{\partial o_p}{\partial w_i} \right) \\
&= - \sum_p (t_p - o_p) \frac{\partial o_p}{\partial net_p} \frac{\partial net_p}{\partial w_i}
\end{aligned}
$$

But we know:

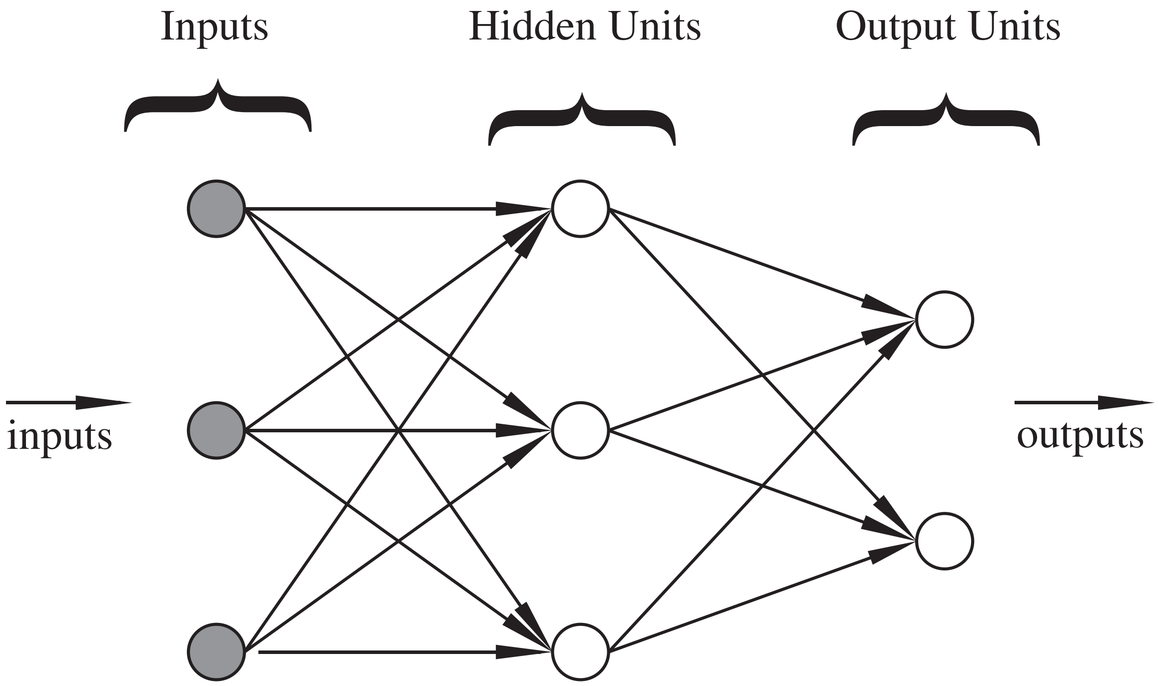
$$
\frac{\partial o_p}{\partial net_p} = \frac{\partial \sigma(net_p)}{\partial net_p} = o_p(1 - o_p)
$$

$$
\frac{\partial net_p}{\partial w_i} = \frac{\partial (\vec{w} \cdot \vec{x}_p)}{\partial w_i} = x_{i,p}
$$

So:

$$
\frac{\partial E}{\partial w_i} = - \sum_{p \in D} (t_p - o_p) o_p (1 - o_p) x_{i,p}
$$

Multi-layer feedforward network



Backpropagation Algorithm for Sigmoid Units

Initialize all weights to small random numbers.

Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k
  $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{1}$$
  3. For each hidden unit h
  $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k \tag{2}$$
  4. Update each network weight $w_{i,j}$
  $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$
  where
  $$\Delta w_{i,j} = \eta \delta_j x_{i,j} \tag{3}$$

More on Backpropagation

- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight *momentum* α

$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

- Minimizes error over *training* examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations $\rightarrow$ slow!
- Using network after training is very fast

Overfitting in Backpropagation

- is a BIG problem, because of large number of weights
- handle it by stopping training after n iterations
- How can we estimate optimal value for n?
  - Use train and test sets
  - Use k-fold cross validation

Expressive Capabilities of Networks

- Can approximate any *continous* function with 3-layer feedforward net [Cybenko, 1989], [Hornik et al., 1989]
  - but might require arbitrarily large hidden layer
  - requires exponential number of hidden units in case of Boolean functions
- Can approximate any function with 4-layer feedforward net [Cybenko, 1988]

Hypothesis space search in Backpropagation

Hypothesis space

- Continuous rather than discrete
- Gradient descent is general search method for continuous H

Inductive bias

- Interpolation, smoothing
- Preference for nets with small weights

ALVINN Example